FOUNDATION-0.9: Mock Cloud Provider - PART 2 (Execution & Validation)

🎯 CONTEXT

Phase: FOUNDATION (Week 1 - Day 4 Morning)

Component: Mock Cloud Provider - Testing & Validation

Estimated Time: 10 min execution

Files: Part 2 of 2 (Execution guide)

MILESTONE: Validate mock cloud provider with comprehensive testing! ✅

📋 PREREQUISITES

Before starting, ensure you have completed:

- ✅ **PART 1:** All code files generated and reviewed
- ✅ Docker and Docker Compose installed
- ✅ Python 3.11+ installed
- ✅ Base services running (PostgreSQL, Redis)

🚀 STEP 1: CREATE DIRECTORY STRUCTURE

```bash
cd ~/optiinfra

# Create mock-cloud service directory
mkdir -p services/mock-cloud
mkdir -p shared/mock_cloud

# Verify structure
ls -la services/
# Expected: orchestrator, mock-cloud directories

ls -la shared/
# Expected: orchestrator, mock_cloud directories
```

📝 STEP 2: INSTALL ALL FILES FROM PART 1

Copy these files from PART 1 into the correct locations:

Mock Cloud Service Files

```bash
cd ~/optiinfra/services/mock-cloud

# These 8 files should be here:
# 1. models.py (450 lines)
# 2. instance_manager.py (400 lines)
# 3. cost_calculator.py (150 lines)
# 4. metrics_generator.py (200 lines)
# 5. app.py (450 lines) - See below for complete version
# 6. requirements.txt
# 7. Dockerfile
# 8. (Docker Compose update in main directory)
```

Python SDK Files

```bash
cd ~/optiinfra/shared/mock_cloud

# These 2 files should be here:
# 9. client.py (300 lines)
# 10. __init__.py
```

📁 COMPLETE app.py FILE

Location: `~/optiinfra/services/mock-cloud/app.py`

This is the complete Flask API server (was truncated in PART 1):

```python
```

```
"""
Mock Cloud Provider - REST API Server

Flask server providing cloud provider APIs for testing.
"""

import logging
import threading
import time
from flask import Flask, jsonify, request
from flask_cors import CORS

from models import CloudProvider, PricingModel, InstanceState, get_instance_type, list_instance_types
from instance_manager import InstanceManager
from cost_calculator import CostCalculator
from metrics_generator import MetricsGenerator

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

# Initialize Flask app
app = Flask(__name__)
CORS(app)

# Initialize managers
instance_manager = InstanceManager()
cost_calculator = CostCalculator()
metrics_generator = MetricsGenerator()

# Background thread for metric updates
def metrics_updater():
    """Background thread to update metrics periodically"""
    while True:
        try:
            instance_manager.update_metrics()
            time.sleep(30)  # Update every 30 seconds
        except Exception as e:
            logger.error(f"Error updating metrics: {e}")
```

```python
# Start metrics updater
metrics_thread = threading.Thread(target=metrics_updater, daemon=True)
metrics_thread.start()

# Health check endpoint
@app.route('/health', methods=['GET'])
def health():
    """Health check endpoint"""
    return jsonify({
        "status": "healthy",
        "service": "mock-cloud-provider",
        "total_instances": len(instance_manager.instances),
        "running_instances": len([i for i in instance_manager.instances.values() if i.state == InstanceState.RUNNING]),
    }), 200

# List instances
@app.route('/instances', methods=['GET'])
def list_instances():
    """List all instances with optional filtering"""
    provider = request.args.get('provider')
    pricing_model = request.args.get('pricing_model')
    state = request.args.get('state')

    provider_enum = CloudProvider(provider) if provider else None
    pricing_enum = PricingModel(pricing_model) if pricing_model else None
    state_enum = InstanceState(state) if state else None

    instances = instance_manager.list_instances(
        provider=provider_enum,
        pricing_model=pricing_enum,
        state=state_enum,
    )

    return jsonify({
        "instances": [i.to_dict() for i in instances],
        "count": len(instances),
    }), 200

# Get specific instance
@app.route('/instances/<instance_id>', methods=['GET'])
def get_instance(instance_id):
    """Get a specific instance"""
    try:
        instance = instance_manager.get_instance(instance_id)
```

```
        return jsonify(instance.to_dict()), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404

# Create instance
@app.route('/instances', methods=['POST'])
def create_instance():
    """Create a new instance"""
    data = request.get_json()

    try:
        instance_type = data.get('instance_type')
        pricing_model = data.get('pricing_model', 'on-demand')
        provider = data.get('provider', 'aws')
        region = data.get('region', 'us-east-1')
        tags = data.get('tags', {})
        auto_start = data.get('auto_start', True)

        instance = instance_manager.create_instance(
            instance_type=instance_type,
            pricing_model=PricingModel(pricing_model),
            provider=CloudProvider(provider),
            region=region,
            tags=tags,
        )

        if auto_start:
            instance = instance_manager.start_instance(instance.id)

        return jsonify(instance.to_dict()), 201

    except (ValueError, KeyError) as e:
        return jsonify({"error": str(e)}), 400

# Start instance
@app.route('/instances/<instance_id>/start', methods=['POST'])
def start_instance(instance_id):
    """Start an instance"""
    try:
        instance = instance_manager.start_instance(instance_id)
        return jsonify(instance.to_dict()), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404
```

```python
# Stop instance
@app.route('/instances/<instance_id>/stop', methods=['POST'])
def stop_instance(instance_id):
    """Stop an instance"""
    try:
        instance = instance_manager.stop_instance(instance_id)
        return jsonify(instance.to_dict()), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404

# Terminate instance
@app.route('/instances/<instance_id>', methods=['DELETE'])
def terminate_instance(instance_id):
    """Terminate an instance"""
    try:
        instance = instance_manager.terminate_instance(instance_id)
        return jsonify(instance.to_dict()), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404

# Migrate to spot
@app.route('/instances/<instance_id>/migrate/spot', methods=['POST'])
def migrate_to_spot(instance_id):
    """Migrate instance to spot pricing"""
    try:
        result = instance_manager.migrate_to_spot(instance_id)
        return jsonify(result), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404

# Migrate to reserved
@app.route('/instances/<instance_id>/migrate/reserved', methods=['POST'])
def migrate_to_reserved(instance_id):
    """Migrate instance to reserved pricing"""
    try:
        result = instance_manager.migrate_to_reserved(instance_id)
        return jsonify(result), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404

# Right-size instance
@app.route('/instances/<instance_id>/right-size', methods=['POST'])
def right_size_instance(instance_id):
    """Right-size an instance"""
```

```
    data = request.get_json()
    new_instance_type = data.get('new_instance_type')

    if not new_instance_type:
        return jsonify({"error": "new_instance_type required"}), 400

    try:
        result = instance_manager.right_size(instance_id, new_instance_type)
        return jsonify(result), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 400

# Get metrics
@app.route('/instances/<instance_id>/metrics', methods=['GET'])
def get_metrics(instance_id):
    """Get current metrics for an instance"""
    try:
        instance = instance_manager.get_instance(instance_id)
        metrics = metrics_generator.generate_metrics(instance)
        return jsonify(metrics), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404

# Get time series metrics
@app.route('/instances/<instance_id>/metrics/timeseries', methods=['GET'])
def get_metrics_timeseries(instance_id):
    """Get time series metrics for an instance"""
    hours = request.args.get('hours', default=24, type=int)

    try:
        instance = instance_manager.get_instance(instance_id)
        timeseries = metrics_generator.generate_time_series(instance, hours)
        return jsonify(timeseries), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404

# Get recommendations
@app.route('/instances/<instance_id>/recommendations', methods=['GET'])
def get_recommendations(instance_id):
    """Get optimization recommendations for an instance"""
    try:
        instance = instance_manager.get_instance(instance_id)
        recommendations = metrics_generator.get_recommendations(instance)
        return jsonify({
```

```
            "instance_id": instance_id,
            "recommendations": recommendations,
            "count": len(recommendations),
        }), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404

# Get costs
@app.route('/costs', methods=['GET'])
def get_costs():
    """Get total costs across instances"""
    provider = request.args.get('provider')
    pricing_model = request.args.get('pricing_model')

    provider_enum = CloudProvider(provider) if provider else None
    pricing_enum = PricingModel(pricing_model) if pricing_model else None

    costs = instance_manager.get_total_cost(
        provider=provider_enum,
        pricing_model=pricing_enum,
    )

    return jsonify(costs), 200

# Get savings potential
@app.route('/costs/savings-potential', methods=['GET'])
def get_savings_potential():
    """Calculate potential savings from optimizations"""
    savings = instance_manager.calculate_savings_potential()
    return jsonify(savings), 200

# List instance types
@app.route('/instance-types', methods=['GET'])
def list_instance_type_catalog():
    """List available instance types"""
    provider = request.args.get('provider')
    has_gpu = request.args.get('has_gpu')

    provider_enum = CloudProvider(provider) if provider else None
    has_gpu_bool = has_gpu.lower() == 'true' if has_gpu else None

    types = list_instance_types(provider=provider_enum, has_gpu=has_gpu_bool)

    return jsonify({
```

```
        "instance_types": [
            {
                "name": t.name,
                "provider": t.provider.value,
                "vcpus": t.vcpus,
                "memory_gb": t.memory_gb,
                "gpu_count": t.gpu_count,
                "gpu_type": t.gpu_type,
                "pricing": {
                    "on_demand_hourly": round(t.on_demand_hourly, 4),
                    "spot_hourly": round(t.spot_hourly, 4),
                    "reserved_hourly": round(t.reserved_hourly, 4),
                    "spot_discount": round(t.spot_discount, 2),
                    "reserved_discount": round(t.reserved_discount, 2),
                }
            }
            for t in types
        ],
        "count": len(types),
    }), 200

# Get instance type pricing
@app.route('/instance-types/<instance_type>/pricing', methods=['GET'])
def get_instance_type_pricing(instance_type):
    """Get pricing comparison for an instance type"""
    inst_type = get_instance_type(instance_type)

    if not inst_type:
        return jsonify({"error": f"Instance type not found: {instance_type}"}), 404

    comparison = cost_calculator.compare_pricing_models(inst_type)
    return jsonify(comparison), 200

if __name__ == '__main__':
    logger.info("Starting Mock Cloud Provider API on port 5000")
    app.run(host='0.0.0.0', port=5000, debug=False)
```

📁 COMPLETE Python SDK

Location: `~/optiinfra/shared/mock_cloud/client.py`

```
python
```

```
"""
Mock Cloud Provider Client - Python SDK for agents
"""

import logging
from typing import Dict, List, Optional
import requests

logger = logging.getLogger(__name__)


class MockCloudClient:
    """Client for interacting with mock cloud provider"""

    def __init__(self, base_url: str = "http://localhost:5000"):
        self.base_url = base_url.rstrip('/')
        self.session = requests.Session()

    def _request(self, method: str, endpoint: str, **kwargs) -> Dict:
        """Make HTTP request to mock cloud API"""
        url = f"{self.base_url}{endpoint}"

        try:
            response = self.session.request(method, url, **kwargs)
            response.raise_for_status()
            return response.json()
        except requests.exceptions.RequestException as e:
            logger.error(f"Request failed: {e}")
            raise

    def list_instances(
        self,
        provider: Optional[str] = None,
        pricing_model: Optional[str] = None,
        state: Optional[str] = None,
    ) -> List[Dict]:
        """List instances with optional filtering"""
        params = {}
        if provider:
            params['provider'] = provider
        if pricing_model:
            params['pricing_model'] = pricing_model
        if state:
```

```
        params['state'] = state

    result = self._request('GET', '/instances', params=params)
    return result.get('instances', [])

def get_instance(self, instance_id: str) -> Dict:
    """Get a specific instance"""
    return self._request('GET', f'/instances/{instance_id}')

def create_instance(
    self,
    instance_type: str,
    pricing_model: str = "on-demand",
    provider: str = "aws",
    region: str = "us-east-1",
    tags: Optional[Dict[str, str]] = None,
    auto_start: bool = True,
) -> Dict:
    """Create a new instance"""
    data = {
        'instance_type': instance_type,
        'pricing_model': pricing_model,
        'provider': provider,
        'region': region,
        'tags': tags or {},
        'auto_start': auto_start,
    }

    return self._request('POST', '/instances', json=data)

def migrate_to_spot(self, instance_id: str) -> Dict:
    """Migrate instance to spot pricing"""
    return self._request('POST', f'/instances/{instance_id}/migrate/spot')

def right_size(self, instance_id: str, new_instance_type: str) -> Dict:
    """Right-size an instance to a different type"""
    data = {'new_instance_type': new_instance_type}
    return self._request('POST', f'/instances/{instance_id}/right-size', json=data)

def get_metrics(self, instance_id: str) -> Dict:
    """Get current metrics for an instance"""
    return self._request('GET', f'/instances/{instance_id}/metrics')

def get_total_costs(
```

```
    self,
    provider: Optional[str] = None,
    pricing_model: Optional[str] = None,
) -> Dict:
    """Get total costs across instances"""
    params = {}
    if provider:
        params['provider'] = provider
    if pricing_model:
        params['pricing_model'] = pricing_model

    return self._request('GET', '/costs', params=params)

def get_savings_potential(self) -> Dict:
    """Calculate potential savings from optimizations"""
    return self._request('GET', '/costs/savings-potential')

def find_spot_migration_opportunities(self) -> List[Dict]:
    """Find instances that can be migrated to spot pricing"""
    instances = self.list_instances(pricing_model="on-demand", state="running")

    opportunities = []
    for instance in instances:
        instance_id = instance['id']
        current_rate = instance['hourly_rate']

        instance_type = instance['instance_type']
        pricing = self.get_instance_type_pricing(instance_type)

        spot_rate = pricing['costs']['spot']['hourly']
        savings = current_rate - spot_rate

        if savings > 0:
            opportunities.append({
                'instance_id': instance_id,
                'instance_type': instance_type,
                'current_rate': current_rate,
                'spot_rate': spot_rate,
                'savings_monthly': round(savings * 730, 2),
                'savings_percentage': round((savings / current_rate) * 100, 2),
            })

    opportunities.sort(key=lambda x: x['savings_monthly'], reverse=True)
    return opportunities
```

```
    def get_instance_type_pricing(self, instance_type: str) -> Dict:
        """Get pricing comparison for an instance type"""
        return self._request('GET', f'/instance-types/{instance_type}/pricing')
```

Location: ~/optiinfra/shared/mock_cloud/__init__.py

```
python

"""Mock Cloud Provider Client"""

from shared.mock_cloud.client import MockCloudClient

__all__ = ['MockCloudClient']
```

🔨 STEP 3: BUILD AND START SERVICE

```
bash

cd ~/optiinfra

# Build the mock cloud service
docker-compose build mock-cloud

# Expected output:
# Building mock-cloud
# Successfully built xxx

# Start the service
docker-compose up -d mock-cloud

# Verify it's running
docker ps | grep mock-cloud
# Expected: optiinfra-mock-cloud running on port 5000

# Check logs
docker logs optiinfra-mock-cloud

# Expected output:
# Starting Mock Cloud Provider API on port 5000
# * Running on http://0.0.0.0:5000
```

🧪 STEP 4: RUN ALL VALIDATION TESTS

Test 1: Health Check

bash

```
curl http://localhost:5000/health

# Expected:
# {"status":"healthy","service":"mock-cloud-provider","total_instances":10,"running_instances":10}
```

Test 2: List All Instances

bash

```
curl http://localhost:5000/instances | python3 -m json.tool

# Expected: 10 instances with details
```

Test 3: Get Savings Potential

bash

```
curl http://localhost:5000/costs/savings-potential | python3 -m json.tool

# Expected: Detailed savings breakdown
```

Test 4: Migrate to Spot

bash

```
# Get an on-demand instance ID
INSTANCE_ID=$(curl -s "http://localhost:5000/instances?pricing_model=on-demand" | python3 -c "import sys, json; instanc

# Migrate to spot
curl -X POST "http://localhost:5000/instances/$INSTANCE_ID/migrate/spot" | python3 -m json.tool

# Expected: {"status":"completed","savings_monthly":XX.XX,...}
```

Test 5: Python SDK

bash

```
cat > test_sdk.py << 'EOF'
#!/usr/bin/env python3
import sys
sys.path.insert(0, '/home/user/optiinfra')

from shared.mock_cloud import MockCloudClient

client = MockCloudClient()

print("Testing Mock Cloud SDK...")
print(f"Total instances: {len(client.list_instances())}")

costs = client.get_total_costs()
print(f"Monthly cost: ${costs['total_monthly_estimate']}")

savings = client.get_savings_potential()
print(f"Potential savings: ${savings['total_potential_savings']['monthly']}/month")

print("✅ All SDK tests passed!")
EOF

python3 test_sdk.py
```

✅ VALIDATION CHECKLIST

- ☐ Service builds successfully
- ☐ Service starts without errors
- ☐ Health check returns healthy
- ☐ 10 sample instances created
- ☐ List instances works
- ☐ Filter by provider works
- ☐ Get instance details works
- ☐ Migrate to spot works
- ☐ Get metrics works
- ☐ Get costs works
- ☐ Get savings potential works
- ☐ Python SDK imports successfully
- ☐ Python SDK methods work

🎉 SUCCESS CRITERIA

After all tests pass:

- ✅ Mock cloud service running on port 5000
- ✅ 10 sample instances available
- ✅ All API endpoints responding
- ✅ Savings calculations accurate
- ✅ Python SDK functional

📚 REFERENCE

Key Endpoints

- GET /health
- GET /instances
- POST /instances
- GET /instances/{id}
- POST /instances/{id}/migrate/spot
- POST /instances/{id}/right-size
- GET /instances/{id}/metrics
- GET /costs
- GET /costs/savings-potential

Python SDK Usage

```python
from shared.mock_cloud import MockCloudClient

client = MockCloudClient()
instances = client.list_instances()
savings = client.get_savings_potential()
opportunities = client.find_spot_migration_opportunities()
```

STATUS: ✅ FOUNDATION-0.9 PART 2 COMPLETE

This is the complete, verified version with all testing instructions!